Chapter 1

Introduction

1.0 Database Management Systems

1. A **database management system** (DBMS), or simply a **database system** (DBS), consists of
   - A collection of interrelated and persistent data (usually referred to as the **database** (DB)).
   - A set of application programs used to access, update and manage that data (which form the data management system (MS)).
2. The goal of a DBMS is to provide an environment that is both **convenient** and **efficient** to use in
   - Retrieving information from the database.
   - Storing information into the database.
3. Databases are usually designed to manage **large** bodies of information. This involves
   - Definition of structures for information storage (data modeling).
   - Provision of mechanisms for the manipulation of information (file and systems structure, query processing).
   - Providing for the safety of information in the database (crash recovery and security).
   - Concurrency control if the system is shared by users.

1.1 Purpose of Database Systems

1. To see why database management systems are necessary, let's look at a typical "file-processing system" supported by a conventional operating system.

   The application is a savings bank:
   - Savings account and customer records are kept in permanent system files.
   - Application programs are written to manipulate files to perform the following tasks:
     - Debit or credit an account.
     - Add a new account.
     - Find an account balance.
     - Generate monthly statements.
2. Development of the system proceeds as follows:
   - New application programs must be written as the need arises.
   - New permanent files are created as required.

- **but** over a long period of time files may be in different formats, and
- Application programs may be in different languages.

3. So we can see there are problems with the straight file-processing approach:
   - Data redundancy and inconsistency
     - Same information may be duplicated in several places.
     - All copies may not be updated properly.
   - Difficulty in accessing data
     - May have to write a new application program to satisfy an unusual request.
     - E.g. find all customers with the same postal code.
     - Could generate this data manually, but a long job...
   - Data isolation
     - Data in different files.
     - Data in different formats.
     - Difficult to write new application programs.
   - Multiple users
     - Want concurrency for faster response time.
     - Need protection for concurrent updates.
     - E.g. two customers withdrawing funds from the same account at the same time — account has $500 in it, and they withdraw $100 and $50. The result could be $350, $400 or $450 if no protection.
   - Security problems
     - Every user of the system should be able to access only the data they are permitted to see.
     - E.g. payroll people only handle employee records, and cannot see customer accounts; tellers only access account data and cannot see payroll data.
     - Difficult to enforce this with application programs.
   - Integrity problems
     - Data may be required to satisfy constraints.
     - E.g. no account balance below $25.00.
     - Again, difficult to enforce or to change constraints with the file-processing approach.

   These problems and others led to the development of **database management systems**.

1.2 Data Abstraction

1. The major purpose of a database system is to provide users with an **abstract view** of the system.

   The system hides certain details of how data is stored and maintained

   Complexity should be hidden from database users.

2. There are several levels of abstraction:

   (a) Physical Level:
   - How the data are stored.
   - E.g. index, B-tree, hashing.
   - Lowest level of abstraction.
   - Complex low-level structures described in detail.

   (b) Conceptual Level:
   - Next highest level of abstraction.

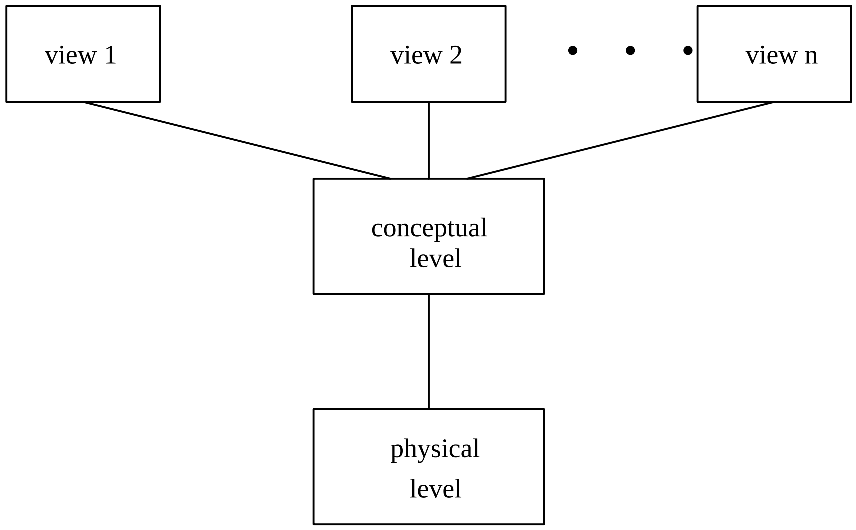

Figure 1.1: The three levels of data abstraction

- Describes *what* data are stored.
- Describes the relationships among data.
- Database administrator level.

(c) View Level:

- Highest level.
- Describes *part* of the database for a particular group of users.
- Can be many different views of a database.
- E.g. tellers in a bank get a view of customer accounts, but not of payroll data.

Fig. 1.1 (figure 1.1 in the text) illustrates the three levels.

1.3 Data Models

1. **Data models** are a collection of conceptual tools for describing data, data relationships, data semantics and data constraints. There are three different groups:

   (a) Object-based Logical Models.

   (b) Record-based Logical Models.

   (c) Physical Data Models.

   We'll look at them in more detail now.

1.3.1 Object-based Logical Models

1. Object-based logical models:

   - Describe data at the conceptual and view levels.
   - Provide fairly flexible structuring capabilities.
   - Allow one to specify data constraints explicitly.
   - Over 30 such models, including
     - Entity-relationship model.

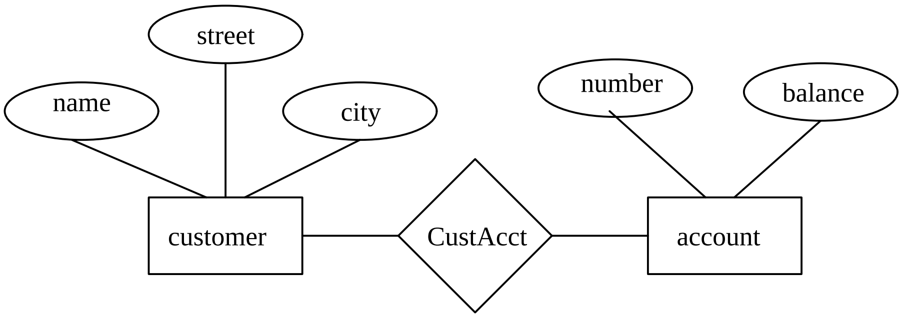

Figure 1.2: A sample E-R diagram.

- Object-oriented model.
- Binary model.
- Semantic data model.
- Infological model.
- Functional data model.

2. At this point, we'll take a closer look at the **entity-relationship (E-R)** and **object-oriented** models.

The E-R Model

1. The entity-relationship model is based on a perception of the world as consisting of a collection of basic **objects** (entities) and **relationships** among these objects.

   - An **entity** is a distinguishable object that exists.
   - Each entity has associated with it a set of **attributes** describing it.
   - E.g. *number* and *balance* for an account entity.
   - A **relationship** is an association among several entities.
   - e.g. A *cust_acct* relationship associates a customer with each account he or she has.
   - The set of all entities or relationships of the same type is called the **entity set** or **relationship set**.
   - Another essential element of the E-R diagram is the **mapping cardinalities**, which express the number of entities to which another entity can be associated via a relationship set.

   We'll see later how well this model works to describe real world situations.

2. The overall logical structure of a database can be expressed graphically by an **E-R diagram**:

   - **rectangles**: represent entity sets.
   - **ellipses**: represent attributes.
   - **diamonds**: represent relationships among entity sets.
   - **lines**: link attributes to entity sets and entity sets to relationships.

   See figure 1.2 for an example.

The Object-Oriented Model

1. The object-oriented model is based on a collection of objects, like the E-R model.

   - An object contains values stored in **instance variables** within the object.
   - Unlike the record-oriented models, these values are themselves objects.

- Thus objects contain objects to an arbitrarily deep level of nesting.
- An object also contains bodies of code that operate on the the object.
- These bodies of code are called **methods**.
- Objects that contain the same types of values and the same methods are grouped into **classes**.
- A class may be viewed as a type definition for objects.
- Analogy: the programming language concept of an abstract data type.
- The only way in which one object can access the data of another object is by invoking the method of that other object.
- This is called **sending a message** to the object.
- Internal parts of the object, the instance variables and method code, are not visible externally.
- Result is two levels of data abstraction.

For example, consider an object representing a bank account.

- The object contains instance variables *number* and *balance*.
- The object contains a method *pay-interest* which adds interest to the balance.
- Under most data models, changing the interest rate entails changing code in application programs.
- In the object-oriented model, this only entails a change within the *pay-interest* method.

2. Unlike entities in the E-R model, each object has its own unique identity, independent of the values it contains:
   - Two objects containing the same values are distinct.
   - Distinction is maintained in physical level by assigning distinct object identifiers.

1.3.2 Record-based Logical Models

1. Record-based logical models:
   - Also describe data at the conceptual and view levels.
   - Unlike object-oriented models, are used to
     - Specify overall logical structure of the database, **and**
     - Provide a higher-level description of the implementation.
   - Named so because the database is structured in fixed-format records of several types.
   - Each record type defines a fixed number of fields, or attributes.
   - Each field is usually of a fixed length (this simplifies the implementation).
   - Record-based models do not include a mechanism for direct representation of code in the database.
   - Separate languages associated with the model are used to express database queries and updates.
   - The three most widely-accepted models are the **relational, network**, and **hierarchical**.
   - This course will concentrate on the **relational** model.
   - The **network** and **hierarchical** models are covered in appendices in the text.

The Relational Model

- Data and relationships are represented by a collection of **tables**.
- Each **table** has a number of columns with unique names, e.g. *customer, account.*
- Figure 1.3 shows a sample relational database.

name	street	city	number
Lowery	Maple	Queens	900
Shiver	North	Bronx	556
Shiver	North	Bronx	647
Hodges	Sidehill	Brooklyn	801
Hodges	Sidehill	Brooklyn	647

name	balance
900	55
556	100000
647	105366
801	10533

Figure 1.3: A sample relational database.

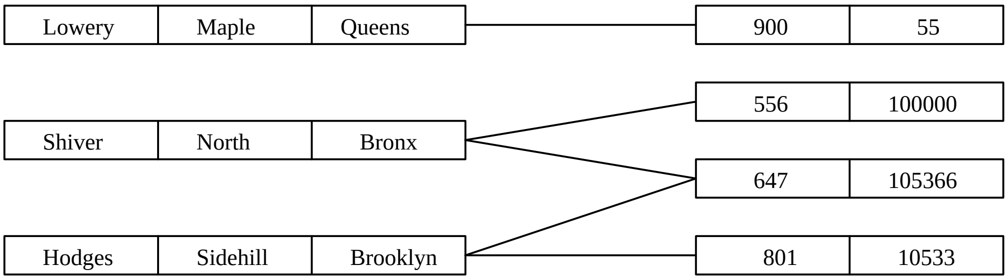

Figure 1.4: A sample network database

The Network Model

- Data are represented by collections of records.
- Relationships among data are represented by links.
- Organization is that of an **arbitrary graph**.
- Figure 1.4 shows a sample network database that is the equivalent of the relational database of Figure 1.3.

The Hierarchical Model

- Similar to the network model.
- Organization of the records is as a collection of **trees**, rather than arbitrary graphs.
- Figure 1.5 shows a sample hierarchical database that is the equivalent of the relational database of Figure 1.3.

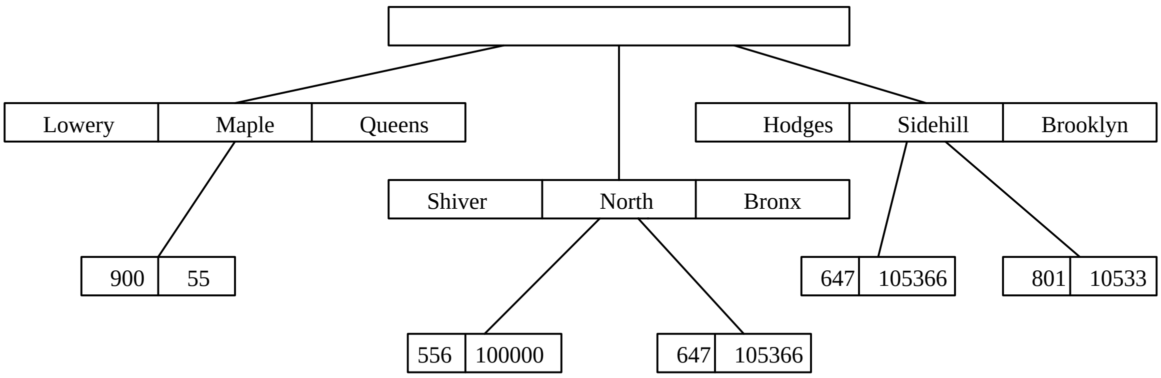

Figure 1.5: A sample hierarchical database

The relational model does not use pointers or links, but relates records by the values they contain. This allows a formal mathematical foundation to be defined.

1.3.3 Physical Data Models

1. Are used to describe data at the lowest level.
2. Very few models, e.g.
   - Unifying model.
   - Frame memory.
3. We will not cover physical models.

1.4 Instances and Schemes

1. Databases change over time.
2. The information in a database at a particular point in time is called an **instance** of the database.
3. The overall design of the database is called the database **scheme**.
4. Analogy with programming languages:
   - Data type definition – scheme
   - Value of a variable – instance
5. There are several schemes, corresponding to levels of abstraction:
   - Physical scheme
   - Conceptual scheme
   - Subscheme (can be many)

1.5 Data Independence

1. The ability to modify a scheme definition in one level without affecting a scheme definition in a higher level is called **data independence**.
2. There are two kinds:
   - **Physical data independence**
     - The ability to modify the physical scheme without causing application programs to be rewritten
     - Modifications at this level are usually to improve performance
   - **Logical data independence**
     - The ability to modify the conceptual scheme without causing application programs to be rewritten
     - Usually done when logical structure of database is altered
3. Logical data independence is harder to achieve as the application programs are usually heavily dependent on the logical structure of the data. An analogy is made to abstract data types in programming languages.

1.6 Data Definition Language (DDL)

1. Used to specify a database scheme as a set of definitions expressed in a DDL
2. DDL statements are compiled, resulting in a set of tables stored in a special file called a **data dictionary** or **data directory**.

3. The data directory contains **metadata** (data about data)
4. The storage structure and access methods used by the database system are specified by a set of definitions in a special type of DDL called a **data storage and definition** language
5. **basic idea:** hide implementation details of the database schemes from the users

1.7 Data Manipulation Language (DML)

1. **Data Manipulation** is:
   - **retrieval** of information from the database
   - **insertion** of new information into the database
   - **deletion** of information in the database
   - **modification** of information in the database
2. A DML is a language which enables users to access and manipulate data.

   The goal is to provide efficient human interaction with the system.
3. There are two types of DML:
   - **procedural**: the user specifies *what* data is needed and *how* to get it
   - **nonprocedural**: the user only specifies *what* data is needed
     - Easier for user
     - May not generate code as efficient as that produced by procedural languages
4. A **query language** is a portion of a DML involving information retrieval only. The terms DML and query language are often used synonymously.

1.8 Database Manager

1. The **database manager** is a **program module** which provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
2. Databases typically require lots of storage space (gigabytes). This must be stored on disks. Data is moved between disk and main memory (MM) as needed.
3. The goal of the database system is to **simplify** and **facilitate** access to data. Performance is important. Views provide simplification.
4. **So** the database manager module is responsible for
   - **Interaction with the file manager:** Storing raw data on disk using the file system usually provided by a conventional operating system. The database manager must translate DML statements into low-level file system commands (for storing, retrieving and updating data in the database).
   - **Integrity enforcement:** Checking that updates in the database do not violate consistency constraints (e.g. no bank account balance below $25)
   - **Security enforcement:** Ensuring that users only have access to information they are permitted to see
   - **Backup and recovery:** Detecting failures due to power failure, disk crash, software errors, etc., and restoring the database to its state before the failure
   - **Concurrency control:** Preserving data consistency when there are concurrent users.
5. Some small database systems may miss some of these features, resulting in simpler database managers. (For example, no concurrency is required on a PC running MS-DOS.) These features are necessary on larger systems.

1.9 Database Administrator

1. The **database administrator** is a **person** having central control over data and programs accessing that data. Duties of the database administrator include:

    - **Scheme definition:** the creation of the original database scheme. This involves writing a set of definitions in a DDL (data storage and definition language), compiled by the DDL compiler into a set of tables stored in the data dictionary.
    - **Storage structure and access method definition:** writing a set of definitions translated by the data storage and definition language compiler
    - **Scheme and physical organization modification:** writing a set of definitions used by the DDL compiler to generate modifications to appropriate internal system tables (e.g. data dictionary). This is done rarely, but sometimes the database scheme or physical organization must be modified.
    - **Granting of authorization for data access:** granting different types of authorization for data access to various users
    - **Integrity constraint specification:** generating integrity constraints. These are consulted by the database manager module whenever updates occur.

1.10 Database Users

1. The **database users** fall into several categories:

    - **Application programmers** are computer professionals interacting with the system through DML calls embedded in a program written in a host language (e.g. C, PL/1, Pascal).
        - These programs are called **application programs**.
        - The **DML precompiler** converts DML calls (prefaced by a special character like $, #, etc.) to normal procedure calls in a host language.
        - The host language compiler then generates the object code.
        - Some special types of programming languages combine Pascal-like control structures with control structures for the manipulation of a database.
        - These are sometimes called **fourth-generation languages**.
        - They often include features to help generate forms and display data.
    - **Sophisticated users** interact with the system without writing programs.
        - They form requests by writing queries in a database query language.
        - These are submitted to a **query processor** that breaks a DML statement down into instructions for the database manager module.
    - **Specialized users** are sophisticated users writing special database application programs. These may be CADD systems, knowledge-based and expert systems, complex data systems (audio/video), etc.
    - **Naive users** are unsophisticated users who interact with the system by using permanent application programs (e.g. automated teller machine).

1.11 Overall System Structure

1. Database systems are partitioned into modules for different functions. Some functions (e.g. file systems) may be provided by the operating system.

2. Components include:

    - **File manager** manages allocation of disk space and data structures used to represent information on disk.

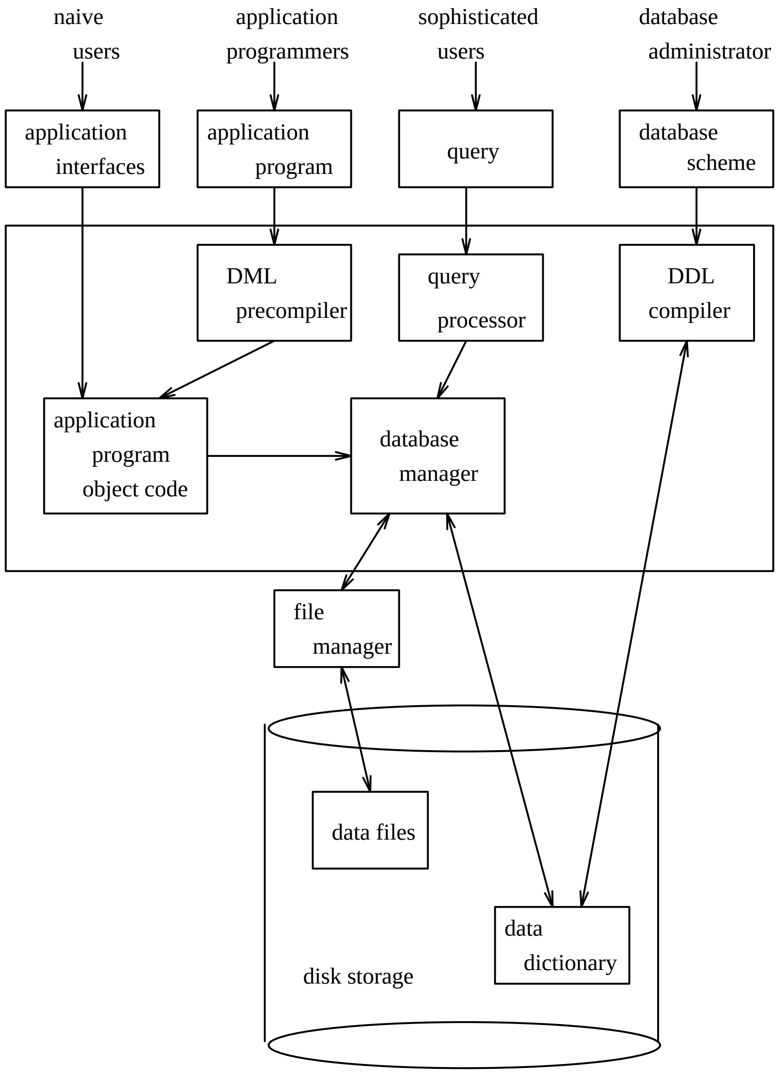

Figure 1.6: Database system structure.

- **Database manager**: The interface between low-level data and application programs and queries.
- **Query processor** translates statements in a query language into low-level instructions the database manager understands. (May also attempt to find an equivalent but more efficient form.)
- **DML precompiler** converts DML statements embedded in an application program to normal procedure calls in a host language. The precompiler interacts with the query processor.
- **DDL compiler** converts DDL statements to a set of tables containing metadata stored in a data dictionary.

In addition, several data structures are required for physical system implementation:

- **Data files:** store the database itself.
- **Data dictionary:** stores information about the structure of the database. It is used **heavily**. Great emphasis should be placed on developing a good design and efficient implementation of the dictionary.
- **Indices:** provide fast access to data items holding particular values.

3. Figure 1.6 shows these components.